## Question 1 (20 marks)

Short Answer Questions

1. Socket Programming – Describe in your words the operating principles, and the pros and cons of TCP and UDP (3 marks) and provide one good example usage for each protocol (2 marks).
2. Kernel Mode v.s. User Mode – Describe the purpose and the operation principles of the kernel mode (2 marks) and user mode (2 marks), and how they are supported in Linux (1 mark)
3. OS Service and API – Describe the purpose and operating principle of an OS service / API (2 marks), and provide three example services that an OS would provide (3 marks).
4. Multiprocessing / Multithreading – Describe the differences between threads and processes (3 marks), and what are the available options to enable the inter-thread communication (1 mark) and inter-process communication (1 mark)

## Question 2 (15 marks)

Write a bash shell script that encapsulate the following simple C program, and provide a menu-based UI that automates the program and continue accepting user commands for repeated program invocation until a “QUIT” command is provided.

The C program “Multiplication Table” is implemented as the following:

#include <stdio.h>

#include <stdlib.h>

int main (int argc, char \*argv[]){

FILE \*fp;

int x;

int multiplier;

char \*fname;

if (argc != 3) {

// need a multiplier a file name to invoke this program properly!

return 1;

}

multiplier = atoi (argv[1]);

fname = argv[2];

fp = fopen (fname, “w”);

if (fp == NULL){

//Invalid file name!!

return 2;

}

for (x = 1; x <= 10; x++) {

fprintf (fp, “%4d x %2d = %d\n”, multiplier, x, multiplier \* x);

}

fclose(fp);

return 0;

}

Your shell script must prompt the end user for the multiplier and the output file name for the desired multiplication table, and verify that the return result indicates success.

* If successful, the shell script must print the contents of the output file, and delete the file afterwards.
* If an error code is received, display the appropriate error message based on the two possible error codes.
* Your shell script must repeatedly accept user command in the format of   
  MULTIPLIER FILENAME after every command execution.
* Your shell script will only terminate if “Quit” command is entered by the user.

Marks will be awarded based on:

* (3 marks) The quality of implementation, including comments, appropriate use of whitespace for readability
* (4 marks) Quality of user interface, including clear prompting messages, error message display, help information, etc.
* (5 marks) Appropriate use of shell script decision / loop constructs
* (3 marks) Bug-free – up to 2 marks will be deducted for coding and design errors.

## Question 3 (15 marks)

Given the following C code implementing a multithreading process, please answer the following questions:

* Produce a step-by-step description 1) the thread creation 2) the respective execution sequences of the two threads, 3) which parts of the program are shared between the two threads, and 4) which parts of the program are separately created for each thread. Support your explanations with diagram illustrations wherever applicable (6 marks).
* Can this code potentially produce a race condition? If yes, explain how the race condition can occur (4 marks).
* Revise this code using the mutex lock from the pthread library to eliminate the race condition. (5 mark)

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <unistd.h>

pthread\_t tid[2];

static int sharedNumber = 666;

void\* myFunction(void \*arg)

{

pthread\_t id = pthread\_self();

if(pthread\_equal(id, tid[0]))

while(sharedNumber < 100000)

sharedNumber++;

else

{

sharedNumber++;

sharedNumber %= 10;

sharedNumber += 5;

sharedNumber--;

}

return NULL;

}

// Continue to next page

int main(void)

{

for(int i = 0; i < 2; i++)

{

if(pthread\_create(&tid[i], NULL, &myFunction, NULL) == 0)

printf(“Thread %d creation failed!\n”, i);

else

printf(“Thread %d successfully created!\n”, i);

}

sleep(3);

return 0;

}

## Question 4 (10 marks)

The following C code is a re-implementation of question 4 in attempt to get rid of the race condition by forking a child process instead. Please answer the following questions:

* Produce a step-by-step description 1) the process creation 2) the respective execution sequences of the two processes, 3) which parts of the program are shared between processes, and 4) which parts of the program are separately created for each process. Support your explanations with diagram illustrations wherever applicable (5 marks).
* Can this code still produce a race condition? If yes, explain how the race condition can occur. If no, explain how this program addresses the race condition seen in question 4 (3 marks).
* Are the two processes independent or cooperative? If cooperative, what variables are shared between the two processes? (1 mark).
* Does this code 100% replicate the functionalities of the code in Question 4 without the race condition? If not, what is missing? (1 mark)

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <string.h>

#include <unistd.h>

static int sharedNumber = 666;

int main(void)

{

if(fork() == 0)

{

while(sharedNumber < 100000)

sharedNumber++;

exit(EXIT\_SUCCESS);

}

else

{

sharedNumber++;

sharedNumber %= 10;

sharedNumber += 5;

sharedNumber--;

wait(NULL);

}

return 0;

}